National Aeronautics and Space Administration
Jet Propulsion Laboratory
California Institute of Technology

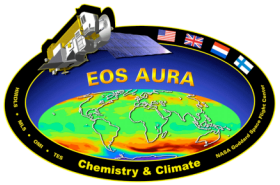
Aura Microwave Limb Sounder

Coding Guidelines for EOS MLS Science Software

David Cuddy

June 15, 2005

Jet Propulsion Laboratory
California Institute of Technology



Agenda

- Introduction
- Background
- Standards
- Specific extensions and rules
- Coding templates
- Naming conventions
- Configuration Management
- Error handling
- Styles
- Summary

Introduction

- EOS MLS (Microwave Limb Sounder) is 1 of 4 instruments on EOS Aura spacecraft – July 15, 2004 launch
- Dr. Joe Waters (JPL) Principal Investigator
- Science Objectives are to improve understanding and assessment of:
  - Stratospheric ozone depletion and chemistry
  - Upper tropospheric ozone distribution and chemistry
  - Climate change and variability
- Measurements
  - Thermal emission from 1 to 100 km altitude in five millimeter and submillimeter wavelength bands
  - Global distribution of O3, ClO, H2O, SO2, CO, N2O, HNO3, HCl, HOCl, BrO, OH, HO2, HCN, temperature, geopotential height, and cirrus ice
  - All measurements are performed simultaneously and continuously, both day and night, and can be made even in the presence of dense cirrus clouds and stratospheric aerosols

Background

- GES-DAAC provides the data archive and distribution
- Science Investigator-led Processing System (SIPS) performs the science data processing
- Science Software (PGEs) development occurs at the MLS SCF by the EOS MLS Science Team
  - All PGEs run in a Linux environment
  - Level 2 PGE runs on a large (364 nodes) cluster using PVM for inter-processor control
- EOS MLS Science Team selected Fortran 95 language for algorithm development and production software
- EOS MLS Science Team adopted a set of guidelines for Fortran 95
- Examined “European Standards for Writing and Documenting Exchangeable Fortran 90 Code” from UK Met Office

Standards

- Fortran 90 includes the complete ANSI standard Fortran 77 plus new features:
  - free source form
  - modern control structures (DO construct, with CYCLE and EXIT options and the control part of the DO can be conventional iteration, WHILE or no control clause, and CASE construct)
  - specification of numeric precision
  - whole array processing
  - dynamic behavior, including allocate, deallocate, pointers, recursion
  - user defined data types
  - Modules
  - operator overloading and generic procedures
- Fortran 95 put some of the Fortran 77 features on the obsolete list plus added some major features:
  - FORALL statement and construct
  - pure and elemental user defined subprograms
  - initial association status for pointers
  - default initialization of derived type objects

Specific extensions and rules

- Fortran 90/95 allow statements to be written in different syntax forms, but MLS guidelines limit the variety for the below
  - CHARACTER declaration
  - Editing descriptor (ES2.d vs 1PEw.d)
  - IMPLICIT statement – only IMPLICIT NONE is acceptable
  - Logical operators – use `>` rather than `.GT.`
  - Multiple statements – use multiple lines for better readability
  - OPEN, READ, WRITE statements must include IOSTAT argument
  - Allocate statements must include STAT argument
  - STOP must use informative message
  - USE statement must include the ONLY clause
  - POINTERS – initialized with "`=> NULL ()`" in declarations

Coding Templates

- Templates are provided for Program, Module, Function and Subroutine
- Templates provide uniformity in placement of the various statements
  - Copyright statement
  - Identifier
  - Description of function/subroutine
  - USE statements
  - PUBLIC/PRIVATE
  - Declarations
  - Procedure code
- Templates provide reminders for required statements such as “IMPLICIT NONE”
- Templates provide an implied style

Naming Conventions

- Keywords are in upper case
- All intrinsic functions and subroutines are in upper case
- Constants are in upper case and preceded by the small letter **c**
- Variables are in lower case except as follows:
  - Variables with two or more “words”, capitalize the first letter of each word
  - Begin scalar variables with lower case
  - Begin array variables with upper case
  - Begin derived type names with a capital letter and end with _T suffix
  - “Object” names begin with a capital letter and end with a Obj suffix
- Program, Module, Subroutine, and Function Names use same format as array names

Configuration Management

- MLS elected to use CVS/RCS
- Near the beginning of the source code, insert as the first data declaration the RCS Identification Information
  - Name of file, version number, last date/time of modification, and id of person who made the last modification
  - During the “checkout” process, CVS/RCS uses this special character string to insert the RCS ID information
- At the end of each source code, wrap up with the comment string:
  - `"! $Log: $"`
  - Recognized by CVS/RCS as the placeholder for all the RCS log messages
  - During the checkout process, RCS lists these messages as comments beginning with the newest message to the oldest message

Error Handling

- MLSMessageModule
  - Provides utilities to handle messages from the PGEs at four severity levels
    - Debug – used only during development and testing
    - Info – routine informative messages used during production
    - Warning – messages for conditions that are unusual but not error
    - Error – conditions that require halting of execution
  - Avoids the use of `print` and `write` for output messages by user
  - Provides a prefix for all messages
  - Controls the logical unit used for messages
  - Suppression of message levels is possible
  - Limiting the number of printed identical messages is possible

Styles

- Indenting: use of 3 spaces per indentation
  - PROGRAM, CONTAINS, and END PROGRAM at the same indentation
  - FUNCTION, END FUNCTION, SUBROUTINE, CONTAINS, and END SUBROUTINE at the same indentation
  - IF, DO, and CASE use indentation for their contents
  - Tab is not recommended in the code, but allowed in comments
- Use of blank spaces to align equal signs for better readability
- Alphabetical ordering in the declaration in addition to grouping of variables by usage
  - Apply alphabetical ordering to module procedures
- Comments explaining usage of every variable and component
  - “Bookmark” comment above each procedure
- Section headers
  - Put all public declarations prior to private declarations
  - Put all public module procedures before private module procedures
  - Place section headers before parameter, type and variable declarations
  - Place one before the executable statements in a procedure

Summary

- Standards are good, but tailoring the use of the standards to a limited subset is better
- MLS placed restrictions on the standard to maintain clarity, consistency, and accuracy in the software development
- Shield code from I/O primitives
- Templates provide a good jumping off point
- Uniformity from naming conventions provides for greater readability and therefore greater ease of maintenance
- Configuration management needs to be an integrated part of coding standard
- Common message handling produces uniformity of behavior of the PGEs and uniformity amongst code writers
- Style for writing code is as important as style requirements for writing of papers or PowerPoint presentations
- Consistent style reduces program maintenance and allows the code writers and readers to easily read each other's code